Testing – Systematic Code Coverage Techniques

By Linda Westfall

the WestfallTeam
Partnering for Software Excellence

www.westfallteam.com

When performing structural testing of the code, the term *paths* refer to control flow sequences through the internal structure of the software. There are typically many possible paths between the entry and exit of a typical software application. Every if-then or if-than-else decision doubles the number of potential paths, every case statement multiplies the number of potential paths by the number of cases, and every loop multiplies the number of potential paths by the number of different iteration values possible for the loop. For example, a software unit with a loop that can be iterated from one to100 times has 100 possible paths (once through the loop, twice through the loop, and so on, up to 100 times through the loop). Add an if-than-else statement inside that loop and that increases the number of paths to 200. Add a case statement with four possible choices inside the loop as well and there are 800 possible paths. Moving from the individual unit to the integration level, if this unit with 800 paths is integrated with a unit that only has two sequential if-than-else statements (four paths), there are now 3200 paths through these two units in combination. Since there are rarely enough resources to test every path through a complex software application or even a complex individual unit, a tester can uses white-box logic coverage techniques to systematically select the tests that are the most likely to help identify the yet undiscovered, relevant defect.

To demonstrate the different white-box logic coverage techniques of statement, decision, and condition coverage, the piece of nonsense code shown in Figure 1 will be used.

```
A = 300
if B > 40 and C < 100 then A = 1000
if B < 60 and C < 20 then A = 10
print A
```

Figure 1: Code with the Input Variables B and C

Statement Coverage: A *statement,* also called a *line,* is an instruction or a series of instructions that a computer carries out. *Statement coverage,* also called *line coverage,* is the extent that a given software unit/component's statements are exercised by a set of tests. Statement coverage is the least rigorous type of code coverage technique. To have complete statement coverage, each statement must be executed at least once. Table 1 illustrates that it only takes one test case to have statement coverage of the code in Figure 1. As long as input variables *B* and *C* are selected so that both decisions in this code are true, every statement is executed.

Test Case #	Inputs		Expected Output
	B	C	A
1	> 40 and < 60	<20	10

Table 1: Statement Coverage - Example

Decision Coverage: A *decision* determines the branch path that the code takes. To have *decision coverage,* also called *branch coverage,* each statement is executed at least once and each decision takes all possible outcomes at least once. For example, if the decision is a Boolean expression, decision coverage requires test cases for both the true and false branches. If the decision is a case statement, decision coverage requires test cases that take each case branch. If decision coverage exists, then statement coverage also exists. Table 2 illustrates the test cases needed to have decision coverage of the code in Figure 1. As illustrated in Figure 2, the test case #2 results in the first decision being true and the second decision being false. Test case #3 results in the first decision being false and the second decision being true. Thus, these two test cases in combination provide decision coverage because the true and false paths are taken out of each decision.

Test Case #	Inputs		Expected Output
	B	C	A
2	>= 60	<20	1000
3	<= 40	<20	10

Table 2: Decision Coverage - Example

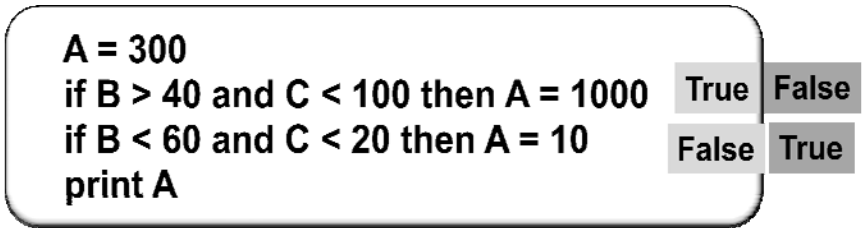

Figure 2: Decision Coverage Test Case Results – Test Cases 2 & 3

Condition Coverage: A *condition* is a state that a decision is based on. To have *condition coverage,* each statement is executed at least once and each condition in a decision takes all possible outcomes at least once. If decision coverage exists, then statement coverage also exists. For the code in Figure 1 there are three conditions that the input variable B can have:

1. $B \leq 40$
2. $40 < B < 60$
3. $B \geq 60$.

There are also three conditions that input variable C can have:

1. $C < 20$
2. $20 \leq C < 100$
3. $C \geq 100$.

Table 3 illustrates one choice of test cases that combines these into condition coverage of the code in Figure 1.

Test Case #	Inputs		Expected Output
	B	C	A
4	>= 60	<20	1000
5	<= 40	>= 20 and <100	300
6	> 40 and < 60	>= 100	300

Table 3: Condition Coverage - Example

Note that condition coverage does not always imply decision coverage. For example, as illustrates in Figure 3, in the set of test cases in Table 3, test case 4 results in the first decision being true and the second decision being false. Test cases 5 and 6 result in both decisions being false. Therefore, decision coverage has not been achieved because the true path has not been taken out of the second decision.

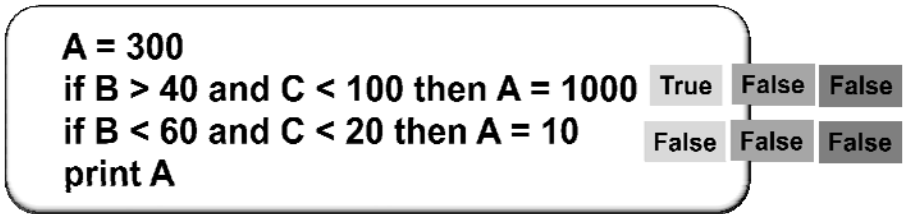

Figure 3: Condition Coverage Test Case Results – Test Cases 4, 5 & 6

Condition and Decision Coverage: The next level of rigor is to have condition and decision coverage where each statement is executed at least once, each decision takes all possible outcomes at least once, and each condition in a decision takes all possible outcomes at least once. If decision/condition coverage exists, condition coverage, decision coverage, and statement coverage also all exist. Table 4 and Figure 4 illustrate one choice of test cases that provides decision/ condition coverage of the code in Figure 1.

Test Case #	Inputs		Expected Output
	B	C	A
7	>= 60	>=20 and < 100	1000
8	> 40 and < 60	<20	10
9	<= 40	>= 100	300

Table 4: Condition & Decision Coverage - Example

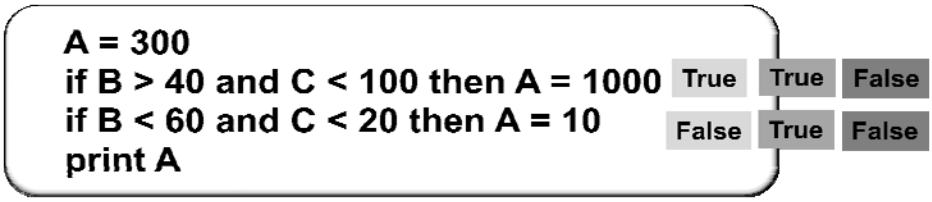

Figure 4: Condition & Decision Coverage Test Case Results – Test Cases 7, 8 & 9

Multiple Condition Coverage: To have *multiple condition coverage*, each statement is executed at least once and all possible combinations of condition outcomes in each decision occur at least once. Multiple condition coverage always results in condition, decision, and statement coverage as well. Multiple condition coverage is the most rigorous type of structural coverage testing. Table 5 illustrates a choice of test cases that provides multiple condition coverage of the code in Figure 1.

Test Case #	Inputs		Expected Output
	B	C	A
1	<= 40	<20	10
2	<= 40	>= 20 and <100	300
3	<= 40	>= 100	300
4	> 40 and <60	<20	10
5	> 40 and <60	>= 20 and <100	1000
6	> 40 and <60	>= 100	300
7	>= 60	<20	1000
8	>= 60	>= 20 and <100	1000
9	>= 60	>= 100	300

Which Type of Code Coverage to Choose: To answer to this question, the tester must consider the risk of the code under test. When selecting the level of coverage, the tester should consider the probability that the code under test has one or more yet undiscovered defects that will escape testing and cause field failures and the cost of those field failures (e.g., cost to the customer/user, cost to correct the defect, cost of propagating the correction back into the field, cost of potential penalty or litigation, cost of lost reputation, cost of lost sale and so on). The higher the probability that defects will escape and cause expensive field failures, the more rigorous level of coverage that should be used. Of course, the tradeoff is the more rigorous the coverage, the longer it will take and the more it will cost to perform the testing.

Without at least statement coverage, part of the code is not tested at all. If that untested code contains a defect, there is no opportunity to identify that defect during test. Many testers would consider decision coverage the minimum level coverage necessary and test all code to at least that level of rigor. Testers would then use the higher levels of coverage rigor as appropriate based on the code under test having a higher level of risk.

About the Author: Linda Westfall is president of Westfall Team, Inc. and has more than 35 years of experience in software engineering, software quality and software project management. She is the author of the *Certified Software Quality Engineer Handbook*. Visit her company's web site at www.westfallteam.com or send comments on this article to lwestfall@westfallteam.com.